Measuring performances and footprint of blockchains with BCTMark: a case study on Ethereum smart contracts energy consumption

Dimitri Saingre, Thomas Ledoux, Jean-Marc Menaud

Measuring performances and footprint of blockchains with BCTMark: a case study on Ethereum smart contracts energy consumption

Dimitri Saingre
IMT Atlantique - Inria - LS2N
Nantes, France
dimitri.saingre@imt-atlantique.fr

Thomas Ledoux
IMT Atlantique - Inria - LS2N
Nantes, France
thomas.ledoux@imt-atlantique.fr

Jean-Marc Menaud
IMT Atlantique - Inria - LS2N
Nantes, France
jean-marc.menaud@imt-atlantique.fr

Abstract—**A rich ecosystem of blockchain-based projects has emerged since the introduction of Bitcoin in 2008. New protocols seek to improve the performances of blockchain systems. In particular, the energy consumption of blockchains has been particularly decried. Unfortunately, those new proposals are often evaluated with ad hoc tools and experimental environments. Therefore, reproducibility and comparison of these new contributions with the state of the art of blockchain technologies are complicated. To the best of our knowledge, only a few tools partially address the design of a generic benchmarking of blockchain technologies (e.g., load generation). This paper introduces BCTMark, a generic framework for benchmarking blockchain technologies on an emulated network in a reproducible way. Based on this novel framework, we studied a key aspect of modern blockchains' energy consumption: smart-contract execution. Based on experiments and the analysis of one year of real-world Ethereum transactions, we measured and modeled smart-contracts' energy consumption on Ethereum. Furthermore, this study details how the replication of contract calls execution can impact their energy cost. In particular, we give insights on the energy consumed by smart-contracts on Ethereum over one year.**

Index Terms—**Blockchain, Performance, Evaluation, Benchmarks, Reproducibility, Smart-contracts, Energy consumption.**

I. Introduction

Since the last few years, the industrial and scientific interest in blockchain technologies does not seem to have weakened. In 2020, according to Google Scholar, more than 43 thousand scientific articles related to the term *blockchain* were published. As a result, a vast ecosystem of blockchain-based projects have emerged in many domains such as decentralized social networks [1], government services [2], storage solutions [3], [4] and energy trading [5], [6]. At the same time, the adoption of blockchain technologies keeps growing. Bitcoin and Ethereum, two of the most significant players in the blockchain ecosystem, now have a market capitalization of 809 and 182 Billion USD, respectively.

Despite the potential of blockchain technologies in many areas, technical limitations slow their development as a possible alternative to centralized services. For example, several issues dealing with their scalability [7], [8] or energy cost [9], [10] have been identified. In particular, Bitcoin (one of the main Blockchain representative) have often been decried for the large environmental impact induced by its consensus engine: Proof-of-Work. In parallel, the blockchain community has proposed new consensus systems such as [11], [12] or the introduction of off-chain transactions systems like [13] in order to increase the performances of their system and limit its energy cost.

Those proposals have been mostly evaluated through debates (e.g., in the case of the *Bitcoin Improvement Proposal*[1]) or have used ad hoc evaluations that are often not reproducible (i.e., cannot be run on systems other than the one they have been designed for). We argue that to properly compare the performances of several blockchain systems and quantify the contribution of new proposals regarding performance issues or functionality (e.g., fault tolerance), the blockchain community needs proper tooling for reproducible experiments.

To the best of our knowledge, only a few tools address the issue of blockchains benchmarking (see section VI). These existing frameworks, while promising, do not always manage aspects necessary for rigorous benchmarking like environment deployment (improving reproducibility), collection of resources usage (e.g., CPU and memory consumption) and network emulation (crucial as, for blockchains, network issues have an impact on the diffusion of new blocks). To help research on blockchain performances analysis, we propose a framework enabling reproducible research on the performances (latency, throughput, energy consumption, . . .) of blockchain technologies. BCTMark (**B**lock**C**hain **T**echnologies Bench**mark**ing) is intended to be a framework which can be used to deploy, compare, and evaluate (through various scenarios) any blockchain on different infrastructures. This framework provides an abstraction of the underlying physical infrastructure and can, therefore, be used to deploy easily on any platform that supports the SSH protocol. To demonstrate this flexibility, we have deployed experiments (in section IV) on both a public research cluster (Grid'5000 [14]) with "classical servers" (Dell PowerEdge R630 servers) and

[1]see BIPs 100 to 107 on the evolution of Bitcoin block size and emission rates

a private ”low-power” Raspberry-Pi cluster. We argue that BCTMark covers the fundamental aspects of benchmarking described in [15]:

- **Repeatability**: BCTMark can manage the whole lifecycle of experiments (resources reservation, deployment, load generation, metrics collection,. . .) and can be used to deploy the same experiment on different infrastructures. Experiments results are consistent across different run (see subsection IV-C).
- **Observability**: BCTMark embeds several components to observe both performances and impact of the system under test (CPU consumption, disk and memory usage, . . .)
- **Portability**: BCTMark can be used to compare different blockchain systems or different versions of the same blockchains. Users can write a driver to use this solution to compare their new system to existing ones.
- **Ease of presentation**: BCTMark embeds a Grafana dashboard [16] that can be used to present the results. Metrics are also stored in a time-series database.
- **Realism**: Network capacities (bandwidth, latency, packet loss, . . .) can be described in the deployment topology to emulate real-world deployment.
- **Ease of run**: BCTMark can manage the whole lifecycle of the experiment (from the resources reservations on a given testbed to the metrics collection of the system under test). It makes them easier to run: the same configuration deployment can be shared with other scientists, even on different testbeds. The deployment topology itself (number of peers, network partition and capacities, . . .) can be easily described in YAML, a language commonly used for configuration.

Leveraging this framework (described in section III), we study in section V the energy consumption of decentralized, smart-contract-based applications. Indeed, even if the energy consumption of Proof-of-Work have been well studied [10], novel consensus algorithms have emerged since. As those new algorithms grow in adoption, it becomes crucial to understand the energy consumption of other key aspects of modern blockchains systems. Therefore, section V proposes a novel method to measure and model the energy consumption of smart-contracts deployed on Ethereum. Based on this model, we give insights into the energy consumed by real-world smart-contracts over one year. Our key findings is that even if smart-contract calls are relatively cheap on their own, their replication across the whole network generates an important energy cost.

To sum up, our contribution results in the design and the development of a framework that researchers can use to create experiments on performance and functionality evaluation of blockchains systems. Taking advantage of this new framework, we then focus (through experiments and the analysis of real-world data) on the energy consumption of decentralized applications on Ethereum. We believe this study will help researchers understanding the energy consumption of blockchain-based applications beyond the analysis of their consensus protocol.

This paper is an extension of a conference paper published in the *17th International Conference on Computer Systems and Applications (AICCSA)* [17]. This extended version present richer *background* (section II) and *related work* (section VI) sections. We also present a complete novel case analysis on measuring and modeling the energy consumption of smart-contracts on Ethereum (see section V).

BCTMark’s code is open-source and accessible here: https://gitlab.inria.fr/dsaingre/bctmark.

II. Background on blockchain technologies

A. Overview

A blockchain can be seen as a distributed data structure that allows facts (called *transactions*) to be recorded as blocks[2]. Each block has a link to the previous one (making a ”chain of block,” or *blockchain*). This data structure is distributed among all participants in a peer-to-peer network. This network is maintained by some peers called *miners* (Bitcoin) or *validators* (Ethereum). Those are in charge of transaction validations.

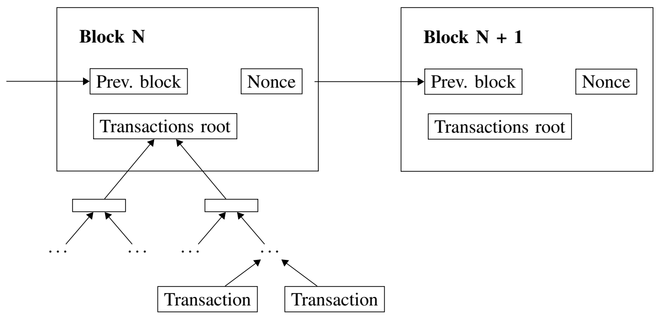

Fig. 1. A schematic view of a classical blockchain ”data-structure”

Validating transactions involves a securing process that can be seen as a *leader election*. The mechanism involved depends on the blockchain system. Probably, the most famous one is called *proof-of-work* (PoW). It involves a ”cryptographic puzzle”. Every block here contains a value called a *nonce*. To validate a block, a miner has to find a value for the nonce, such as the hash value of the whole block is under a certain threshold (called the *difficulty*). This threshold value varies so that, even as the hardware becomes more powerful, the throughput of the entire network remains at about one block per 10 minutes.

Although the *proof-of-work* has been widely used with Bitcoin, it has been quite criticized for its high energy consumption [10]. Alternatives such as *proof-of-stake* (PoS) [11] or *proof-of-elapsed-time* [18] have emerged. Instead of basing its security on computational power, *proof-of-stake* systems rely on the distribution of wealth. In these systems, the probability of validating a block is proportional to the number of coins one owns (in some cases, coins can have a certain weight to avoid having a network led by the richest).

[2]A batch of transactions

B. Decentralized applications with Smart-contracts

Some blockchains, like Ethereum [19], have a concept of *smart-contracts*. Smart-contracts are scripts written in a high-level programming language that can be deployed and executed through network transactions.

Once written, those smart-contracts can be deployed on the network through a transaction containing their compiled code. On Ethereum, transactions without any recipient are used for smart-contract deployment. Once deployed, the smart-contract gets an address like any "normal" accounts[3]. The contract can then be called by sending a transaction to its address containing a compiled version of a function called with desired parameters (if any). Peers receiving the transaction will execute it. On Ethereum, smart-contracts are executed on a specific virtual machine called the *Ethereum Virtual Machine* (EVM).

Smart-contracts offer many computational possibilities and are the backbone of any blockchain-based decentralized applications. As those contracts need to be deterministic (as every peer running the contract needs to produce the same result), they cannot have any side effects outside the blockchain (e.g., they cannot call any Web services). As today, one of the performance limitations of the peer engines executing transactions (and so smart-contracts) is that they execute all the transactions sequentially (and therefore missing the capabilities of multi-core processors). Nonetheless, work (such as [20]) is ongoing in this area.

In Ethereum, transaction emission requires a fee paid by the caller. This fee will be collected by the miner that will include the corresponding transaction in its block. Therefore, it serves as an incentive for miners to include the transaction in their blocks. Transaction fees are calculated as such: $transaction\_fee = gas\_cost \times gas\_price$ where gas_price is defined by the caller and gas_cost equals to the sum of gas cost of each *EVM* instruction called during the smart-contract function execution. As a result, the more computations a smart-contract function does, the more expensive its call will be. Callers can freely define the price (gas_price) they are willing to pay for each gas unit the contract call will consume (gas_cost). This price depends on the market's state: a higher gas price will make the transaction more expensive for the caller to execute, but will serve as a higher incentive for miners to include the transaction in their blocks. Ethereum's Yellow Paper [21] defines the gas cost of each *EVM* instruction. A sample of the cost table can be found in Table I. We can see that every transaction has an inherent cost of 21 000 gas ($G_{transaction}$). Deploying a new smart-contract induces a minimal cost of 53 000 gas (21 000 gas for the transaction itself and 32 000 to create a new account).

As the amount of computation performed by a smart-contract and its cost in gas are correlated, studying the gas cost of smart-contract calls can give us insights on how computation-intensive Ethereum's smart-contracts are.

Name	Value	Description
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{create}	32000	Paid for a CREATE operation.
G_{call}	700	Paid for a CALL operation.
$G_{transaction}$	21000	Paid for every transaction.

TABLE I
SAMPLES OF GAS COST DEFINITION FROM ETHEREUM YELLOW PAPER[21]

However, we cannot infer what functionality these contracts implement only with their cost in gas. To do so (and without analyzing each smart-contract's source code), it is possible to use commonly used contract interfaces.

C. Public vs. Private Blockchains

Blockchain technologies can be divided into two categories: public and private blockchains. Public blockchains, such as Bitcoin and Ethereum (with its Ethash [22] engine), have no identified users. One can join or leave the network at any time without the need for any authorization. Security protocols of those public blockchains need to be enforced to face potential Byzantine faults. Proof-of-work is an example of a consensus system for public blockchains.

Private blockchains have different security models. They aim to identify participants, especially for the block validators. These are designated in the protocol so that no one else can validate the block. These blockchain systems, such as Ethereum (with its *Clique* engine) and Hyperledger Sawtooth (with its *Proof of Elapsed Time* system, based on the *Intel SGX* enclave), have different consensus engines. These engines have better performances (due to the different security models considered) but offer a lower degree of decentralization.

D. Testing the performances of blockchain technologies

Lal and Marijan [23] have produced an extensive survey on the literature regarding Blockchain testing. A large part of the research on Blockchains is focused on testing the performances and correctness of those technologies. Efforts on testing blockchain technologies can be divided into several categories, including Blockchain performances testing, smart-contract testing, and security testing.

1) Blockchains testing: A common approach to evaluate performances of existing blockchains is to deploy them in a private and controlled environment and measure different metrics while running a workload. Such approaches are discussed in subsection VI-A. In [24], Zheng et al. proposed a different approach with a real-time performance monitoring framework. This framework aims to analyze the performances of blockchain peers running on a public network. In this case, the goal is not to analyze performances in a controlled scenario but to study the performances of running peers. Simulators for blockchains (such as BlockSim [25] and DAGSim [26]) have also been proposed. Such simulators help to understand the evolution of performances depending on different system parameters.

[3]The presence of bytecode is the main difference between a contract account and a "human" account

2) Smart-contracts testing: Smart-contracts are an essential part of modern blockchain platforms, as they enable the development of complex services. Smart-contracts developers need to be able to assess the correctness of their contracts due to their immutability aspect (smart-contracts source code cannot be modified once deployed). Several static and dynamic methods have been developed to verify the performances and correctness of contracts in development. Some of those tools and techniques are discussed in subsection VI-B. Moreover, the use of formal method (like with Verx [27]) can help to detect bugs before deployment.

We have illustrated in this section the variety of technologies behind the term *blockchain*. According to Google Scholar, the number of publications concerning the term *blockchain* was 9 510 in 2017, 25 700 in 2018, 33 000 in 2019 and 35 000 in 2020 (at the moment where this paper was written). This trend tends to illustrate a gain of interest in blockchain technologies. However, to the best of our knowledge, only a few tools exist to evaluate emerging blockchain systems. To address this lack, we introduce BCTMark, a framework for benchmarking blockchain technologies.

III. BCTMark

This section presents BCTMark, our solution for benchmarking blockchain technologies. We first introduce how BCTMark can be used to run existing experiments and how developers/scientists can integrate new blockchain systems to be tested. Then, we detail its architecture and underlying components.

A. Usage

From a user's point of view, the workflow of an experiment performed with BCTMark proceeds as described in Figure 2.

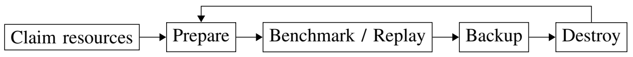

Fig. 2. The experiment workflow of BCTMark

The first step is to *claim* resources on which to deploy the experiment. BCTMark is intended to be portable to manage repeatable experiments. Experiments can be deployed on any infrastructure that supports SSH connections. Some research testbeds (like Grid'5000) require users to book resources before using them. This reservation phase can be addressed by BCTMark. As shown in Listing 1, the deployment topology can be described in a YAML file. This provided example can be used to deploy on a local device 1) an Ethereum network with one bootnode and two peers, 2) one benchmark worker (used to generate loads), 3) a "dashboard" server that hosts both the monitoring stack and the load generator master (that coordinate workers, see subsection III-B for details on load generation). In this case, the *claim* phase will only start the required virtual machines.

Once the infrastructure resources claimed, BCTMark can *prepare* the experiment by deploying the required components (i.e., download and install dependencies, copy configuration files...). For each *role* (see Listing 1), there is a corresponding component to be deployed. The monitoring stack (*dashboard* role) and the benchmarking workers (*bench_worker* role) are common to many experiments. Users can define their roles to deploy their blockchain network. In the example in Listing 1, we need two roles to deploy an Ethereum network: bootnodes[4] and peers.

After deployment, users can run the benchmark themselves. BCTMark provides two possibilities to do this: an ad hoc load generation and a one based on previous traces (for more details on the implementation choices, readers may refer to subsection III-B). Once the benchmark has ran, the results can be backed-up, and the environment destroyed/cleaned for another experiment.

```
deployment:
  vagrant:
    backend: virtualbox
    box: generic/debian10
    resources:
      machines:
        - roles: ["dashboard"]
          flavour: tiny
          number: 1
        - roles: ["ethgethclique:bootnode"]
          flavour: tiny
          number: 1
        - roles: ["ethgethclique:peer"]
          flavour: tiny
          number: 2
        - roles: ["bench_worker"]
          flavour: tiny
          number: 1
```

Listing 1. Configuration example for local deployment with Vagrant

From a developer's point of view, all the following necessary actions must be implemented to integrate a new blockchain to be tested:

- *Deployment*: write a new Ansible playbook (cf. subsection III-B) that specify how to deploy, backup and delete the system;
- *Metric collection*: write a Telegraf plugin (cf. subsection III-B) to gather system-specific metrics (e.g., block emission rate) if not already available through HTTP web services (BCTMark can collect metrics exposed at given HTTP endpoint);
- *Adhoc Load generation*: write functions that correspond to an interaction one can have with the system (e.g., how to send a transaction, how to call a smart-contract, ...);

[4]Bootnodes are peers that have an address known by everyone in the network. New peers can connect to those bootnodes to get the address of other peers in the network

- *Reproducible Load generation*: implement functions to backup transactions (and serialize those) and functions to replay a given serialized transaction.

A developer/researcher would benefit from the design of BCTMark as a framework to easily integrate its new blockchain technology to be tested. Indeed, BCTMark already provides:

- *Deployment*: portability of deployment on several testbeds that support SSH;
- *Network emulation*: latency, bandwidth limits, . . . ;
- *Metric collection*: collection of metrics related to the infrastructure (e.g., CPU usage);
- *Load generation*: distribution of the load to generate among workers.

Only specific interactions with the blockchain to be tested need to be implemented.

B. Architecture

To avoid reinventing the wheel, BCTMark is based on the state-of-the-art industry-proven tools. Altogether they empower researchers, allowing them to provision computing resources, deploy blockchain peers, generate load (based on a history to reproduce or according to a given scenario), and collect metrics relating to peers' performance and energy consumption. The architecture of BCTMark is illustrated, through a UML component diagram, in Figure 3.

Deployment. BCTMark can deploy the entire experiment stack: system under test, monitoring system, and load generators.

Deployment does not require any agent installation on the machines. They are managed through SSH. A *playbook* defines the configuration to be deployed, which takes the form of configuration files in YAML format. Those configuration files make it possible to specify the desired deployment in a relatively explicit, documented, and repeatable way. BCTMark also provides an abstraction layer of the underlying infrastructure. The deployment topology can be described in a relatively high-level point of view, portable on different testbeds. That makes experiments portable on various infrastructures such as Vagrant (local deployment), Grid'5000 and Chameleon.

To manage deployment, BCTMark uses EnosLib [28] (an open-source library to build experimental frameworks) and Ansible [29] (a software that allows to manage deployment of configuration on a cluster). These two components enable self-describing, reproducible deployments.

Metrics management. Metrics about the server (CPU, memory consumption, HDD usage, . . .) and blockchains (number of blocks produced, hashrate, . . .) are collected, stored and displayed by Telegraf [30], InfluxDB [31] and Grafana [16] in time-series, respectively.

Telegraf natively allows the collection of server metrics through many plugins written in Go. New ones can be developed to manage the collection of data on deployed blockchain peers. Current experiments on Ethereum deployment use the HTTP plugin from Telegraf to collect metrics through the Ethereum HTTP API.

Network Emulation. One strength of BCTMark is its ability to describe simply the desired network to emulate. Users can describe in the YAML deployment configuration file several groups of peers and emulate any desired network condition between them. The current characteristics of the network that can be emulated are the percentage of packet loss, network delay, and network rate (i.e., bandwidth). A use case of this feature could be to study the effect of a sudden network partitioning or merge on a blockchain system. Under the hood, BCTMark uses EnosLib that applies the desired network rules using the Linux command TC.

Load generation. BCTMark supports two ways to generate workloads[5]: a *ad hoc* load generation (based on Python scripts) and a load generation based on an history. The first one uses Locust [32], a load generator written in Python. The user needs to specify, through Python methods, any interaction a user can have with the system under test (e.g., sending a transaction to someone or deploying/calling a smart-contract). Locust will then use those methods to generate random loads.

The second way to generate load is based on a provided history. BCTMark can extract the history of a peer in the system and serialize it in a YAML file containing all the transactions. To reproduce the history, it can split the transactions between different workers, create the number of accounts needed to replay it, and let the workers re-run the transactions. This way, we can aim to replay transactions issued from the *mainnet* of a targeted blockchain system.

Energy consumption. BCTMark does not embed any energy monitoring tools. However, as it enables the deployment of experiments on any kind of testbeds, it can be used to deploy systems on clusters where the energy consumption is monitored. We have already tried this by deploying experiments on the SeDuCe [33] cluster (see subsection IV-A). It is part of the Grid'5000 testbed and is monitored with both energy and thermal sensors.

IV. Validation experiments

In this section, we illustrate BCTMark's capabilities through three experiments. The first one demonstrates its capacity to deploy experiments on different testbeds, the second one its capacity to compare two blockchain systems and the third one, its usage for smart-contract performance evaluation. Those experiments use two different testbeds (both having power measuring capacities):

1) A Raspberry-pi 3+ cluster. Each node has a quadcore Cortex-A53 ARMv7 CPU and 1GB of RAM.
2) Grid'5000 [14] Ecotype: A Dell PowerEdge R630 cluster. Each node has two Intel Xeon E5-2630L v4 (Broadwell, 1.80GHz, 10 cores/CPU) CPU and 128 GiB of RAM. Grid'5000 is a large scale public research testbed containing several clusters. Ecotype is one of those clusters, located in Nantes (France).

We evaluated three blockchain systems:

[5]By *workload*, we mean *transactions* to be processed by the system under test

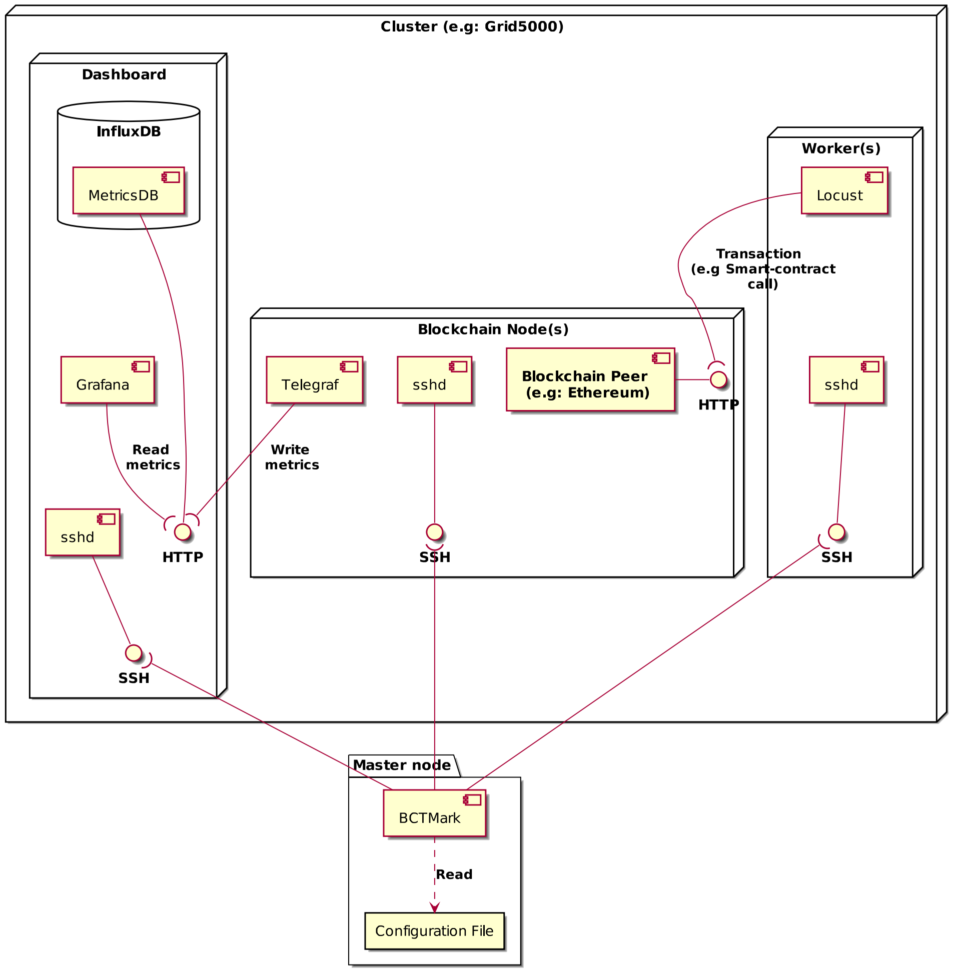

Fig. 3. BCTMark architecture

1) Ethereum Ethash, an implementation of the Proof of Work (PoW) system of Ethereum. It is the default implementation of Ethereum, used in the context of a public blockchain. In this system, every peer can actively participate to block mining.
2) Ethereum Clique, an implementation of the Proof of Authority (PoA) system of Ethereum. PoA is used in the context of a private blockchain. In this system, pre-selected and identified peers can validate blocks one at a time. It does not involve any mining.
3) Hyperledger Fabric. It is also intended for private blockchain. Peers submit transactions to special peers called *orderers*. Orderers are in charge of the ordering process of transactions. Hyperledger Fabric uses a voting-based consensus protocol.

A. Deployment of blockchains on two different testbeds

This experiment illustrates the capabilities of BCTMark to deploy blockchains on different testbeds. We have deployed Ethereum Clique on both Raspberry Pi and Ecotype cluster under three scenarios. The IDLE scenario does not include any load generation. Peer just generate and share empty blocks. The two other scenarios include a load generation of 5 and 50 transactions per second. Load is generated by separated workers and spread randomly across peers. For both experiments, we deployed 12 peers and 6 load generator workers.

Results are presented in Figure 4. The bar plotted on the graph corresponds to the average power usage of every machines in the cluster. The error bar illustrates the standard deviation of power usage.

Those two platforms have different power draw. Power usage on the Dell servers goes from 130.4 to 131.54 watts (0.7% increase) whereas power usage on the Raspberry Pi platform goes from 3.4 to 5.2 watts (44% increase). This result was expected as Raspberry Pi are much more limited than classical ”high performances” Dell servers. This experiment however illustrates that non-mining chains can be installed on low-power platforms like Raspberry Pi. This can be useful in the context of the development of blockchains in IoT / Edge computing. In the context of research on energy consumption, low-power platforms can be useful to illustrate subtle differences in the consumption.

We can, however, note that this conclusion may not be the

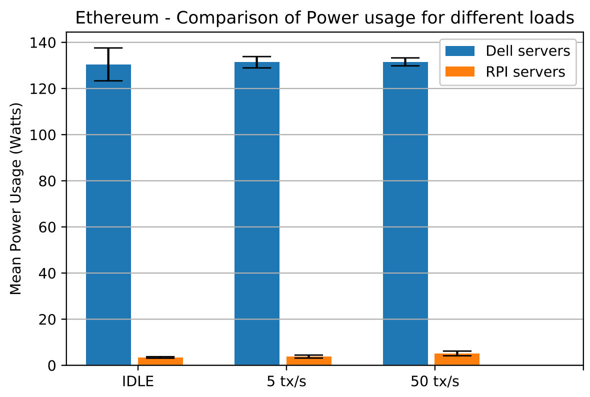

Fig. 4. Comparison of Power Usage for different loads

same for mining systems such as Ethereum Ethash. Indeed, we could not install Ethash on our Raspberry Pi platform due to shortage in memory. The algorithm used by Ethereum Ethash for mining is memory intensive and therefore not suited for low-power platforms with not enough RAM. A solution for this issue could be to set-up both high-performance nodes dedicated to mining and low-power nodes that would only broadcast transactions to the miner's network.

B. Comparison of CPU usage of three blockchain systems

This experiment aims to illustrate the capabilities of BCT-Mark to deploy different blockchain systems. We deployed Hyperledger Fabric, Ethereum Ethash, and Ethereum Clique on the Ecotype cluster under four scenarios: IDLE (no-load generation) and load generation of 5, 50, and 200 transactions per second. The deployed network is composed of a network of 39 peers and three load generator workers. Figure 5 illustrates this experiment. The bar corresponds to the average CPU usage across all machines, whereas the error bar goes from the 10th quartile to the 90th quartile.

We can first notice that the CPU consumption of the Ethash system exceeds the CPU usage of the two others. Moreover, in this deployment, peers only mine blocks using one thread. It could be possible to dedicate more resources for mining, increasing the CPU consumption furthermore. The other two systems have non-mining consensus systems, decreasing the amount of computation needed to secure the network.

The CPU usage of non-mining systems are also more stable than the Ethash system. Figure 6 illustrates the evolution of the CPU usage for Ethash peers during the "200 transactions per second" scenario[6]. The spike at the beginning of the experiment, reaching almost 100% CPU, is due to the construction of the data structure needed by peers to start mining. We can also see that, after this spike, the CPU usage increases over time. This increase may be due to the evolution of the difficulty in mining resulting from the mining competition between peers.

[6]CPU usage values seem to differ from ones in Figure 5 but this visual effect is due to 1) high variance in data and 2) high density in data points that hides lowest values. We can notice the high variance on Figure 5.

On the other hand, in the first three scenarios, the CPU consumption of the two private blockchains is roughly the same. However, at 200 transactions per second, the CPU consumption of the Ethereum Clique network increases from 0.3% to 2.9%. This increase suggests that Hyperledger Fabric could have better performances in the context of a private blockchain. These results about private blockchains are consistent with those shown in the Blockbench paper [34].

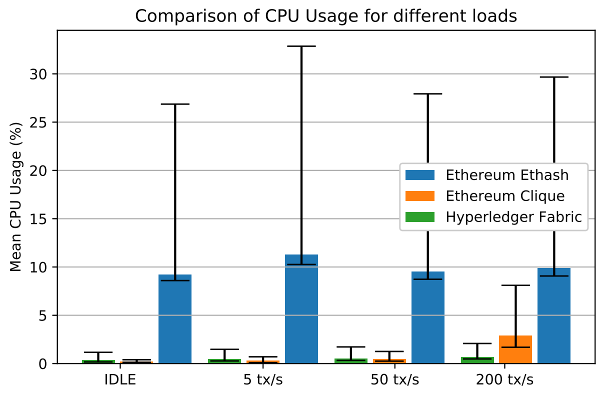

Fig. 5. Comparison of CPU Usage for different loads

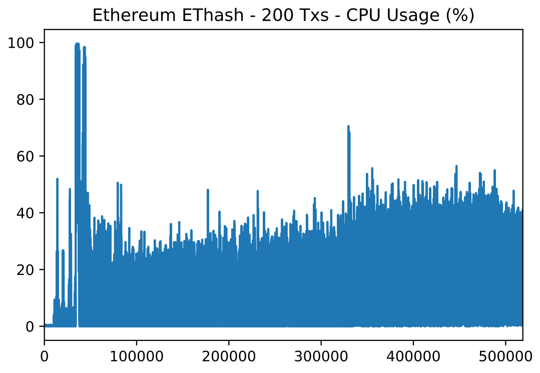

Fig. 6. Evolution of Ethash CPU usage for 200 Txs

C. Experiments Reproducibility

One of the goals behind BCTMark was to enforce reproducibility on blockchain experiments. Reproducibility means that running experiments several times (in similar conditions) should give coherent results. The goal of this section is to illustrate how experiments made with BCTMark can be reproduced.

We reproduced the experiments done in subsection IV-B on Ethereum Clique and Ethash to have data on both public and private blockchain systems (readers can refer to this section to read about the infrastructure used and the deployment topology). Each of the four scenarios has been run six times. For every run, we have recorded the average CPU usage across all machines. The data presented in Table II illustrate the differences in the results we obtained. For instance, the min column illustrates the min CPU average across the six runs. Experiments should show consistent results to be considered reproducible.

These results show that we obtained few differences between the six runs. The standard deviation (column 'Std') remains low across all scenarios. This small difference in results leads us to believe that experiments with BCTMark should produce consistent results. Having exactly the same deployment topology with the same configuration is, in our opinion, the main factor explaining these consistent results. BCTMark allows researchers to share experiments that can be run in the same way by other peers in their community.

Deployment	Min	Max	Mean	Std
Ethereum Clique IDLE	0.113	0.117	0.115	0.002
Ethereum Clique 5 Txs	0.138	0.155	0.145	0.007
Ethereum Clique 50 Txs	0.463	0.526	0.494	0.028
Ethereum Clique 200 Txs	1.682	3.686	2.185	0.843
Ethereum Ethash IDLE	8.751	9.574	9.158	0.304
Ethereum Ethash 5 Txs	9.137	10.169	9.542	0.410
Ethereum Ethash 50 Txs	9.192	11.012	9.945	0.821
Ethereum Ethash 200 Txs	8.934	10.621	9.584	0.630

TABLE II
REPRODUCIBILITY ACROSS SIX RUNS

V. Case study: modeling the energy consumption of smart-contracts in Ethereum

This section leverages BCTMark to measure and model the energy consumption of smart-contracts in Ethereum (one of the leading open source blockchain system). With our work, we aim to initiate studies that goes beyond the measure of the energy consumption of blockchain consensus and take into account the complexity of modern blockchain-based decentralized applications.

A. Preamble - Deploying smart-contracts with BCTMark

The experiment, presented in Figure 7, is intended to illustrate the capabilities of BCTMark for performance analysis of software developed for blockchains. As explained in subsection II-B, blockchains like Ethereum enable developers to write applications through smart-contracts. As detailed in section II, each call to a smart-contract on Ethereum requires a "fee" related to its cost in gas. Gas is a unit related to the computational cost of each instruction in a contract. The more computation there is in a contract, the more expensive for end-users it will be. Moreover, in each mined block, there can be a limit to the sum of each transaction's cost in gas. As a result, there is an incentive for smart-contract developers to control their contract's cost in gas.

To illustrate how implementation design and details can impact the cost of a smart-contract, we implemented three classical sorting algorithms: Quicksort, Bubblesort, and Mergesort. We then have deployed those contracts on a four nodes Ethereum network and generated calls to those contracts. We have measured the cost in gas for each call. For each algorithm, we have generated calls to its sorting function with a random array of integers.

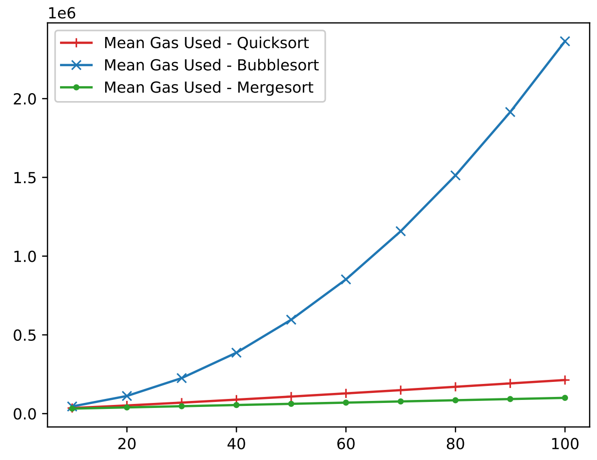

Fig. 7. Cost in gas of three smart-contracts depending on provided input

Figure 7 illustrates the evolution of gas consumption depending on the size of the input array. We can see that the evolution of gas requirements are coherent with the complexity of those three algorithms. Quicksort and Mergesort have the same average complexity of $O(nlog(n))$, whereas Bubblesort has an average complexity of $O(n^2)$. This statement is reassuring as it tends to show that the EVM[7] has been correctly implemented. This simple example demonstrates that BCTMark can be used to study smart-contracts' costs and that its implementation has an impact on smart-contracts possibilities. For the same iso-functionality (here, sorting), the cost of calling the contract will differ depending on the underlying algorithm. For information, at the time this article was written, the Mergesort algorithm would have cost around \$0.55 to sort 100 items (cost of ∼101659 gas). In contrast, the Bubblesort algorithm would have cost around \$11.77 (cost of ∼2168761 gas)[8].

B. Smart-contracts footprint on non-Proof-of-Work systems

The high energy footprint of Proof-of-Work-based blockchain systems has been established several times [10], [35], [9]. However, we did not find any studies focused on the energy consumption of smart-contracts. We aim to illustrate here, through experimentation, how the energy consumption of smart-contracts execution will become more significant in non-Proof-of-Work systems.

To do this, we deployed, with BCTMark, a private Ethereum network on a cluster of power-monitored servers [9]. This cluster of six Dell PowerEdge R640 servers, is equipped with Intel Xeon Gold 5220 18 cores CPU, 96 GiB of memory, 480 GB SSD SATA Micron MTFDDAK480TDN, and 25 Gbps Ethernet connection. Ethereum's peers have been deployed on five servers. The remaining server was used to host the monitoring

[7]Ethereum Virtual Machine, the VM running smart-contracts
[8]Calculated on https://ethgasstation.info with average gas price
[9]Each server is equiped with a sensor that measure its energy consumption

dashboard and database. To illustrate the difference in energy consumption between running a smart-contracts on a Proof-of-Work system and running the same contract on a Proof-of-Authority system, we deployed the Quicksort contract detailed in subsection V-A. For each system (Ethereum Proof-of-Work and Proof-of-Authority), we studied the energy consumption on two cases. In the first, called IDLE, peers are mining empty blocks (no transactions are emitted). In the second, we add a constant load generation consisting of contract calls with a random array of size 1000 to be sorted.

Figure 8 compares the Ethereum Proof-of-Work system's power usage with and without (IDLE) contract calls. In both cases, we can notice a succession of plateaus that corresponds to the mining process's energy consumption. The IDLE network consumes on average 156 Watts, whereas the network executing contract calls consumes on average 160 Watts. The constant execution of contracts represents a 2% increase in its energy consumption.

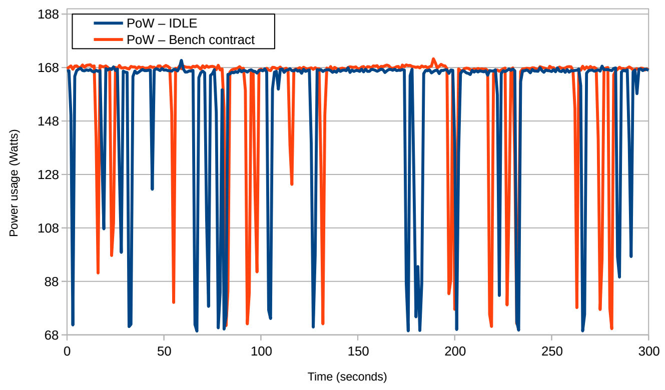

Fig. 8. Comparison power usage with and without contract execution on Ethereum PoW

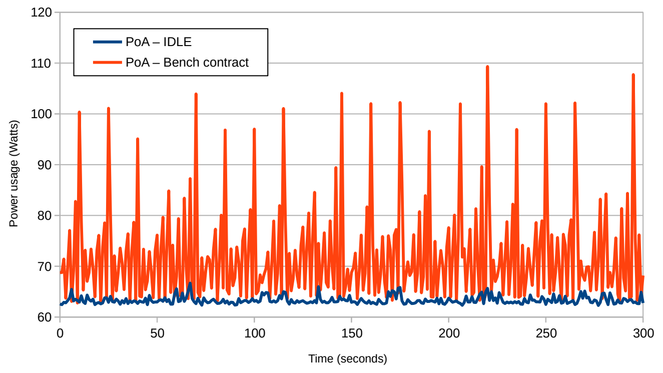

Fig. 9. Comparison power usage with and without contract execution on Ethereum PoS

Figure 9 illustrates the same deployment on an Ethereum network using Proof-of-Authority. The average power usage on the IDLE system is 63.9 Watts, which represents 40% of the power usage of our IDLE Proof-of-Work system. Running calls of our smart-contract raise the power usage of our system to an average of 72.1 Watts. This represents an increase of 13.9%. We observe spikes in the power consumption in the deployment with contract executions. These spikes are due to the processing of those smart-contract calls. No spikes are visible with the Proof-of-Work engine, as the overhead in power consumption is too small there. In addition, no plateau is visible in the Proof-of-Authority deployments because this mechanism does not involve any mining for securing the blockchain.

From these two experiments, we can conclude that smart-contracts execution implies more overhead, in proportion, in systems without Proof-of-Work. As the adoption of non-mining blockchain systems will rise [10], one of the main energy-consuming parts of blockchains will become the actual processing of smart-contract calls and transactions. That is why we estimate that research will focus on reducing the energy footprint of transactions and smart-contract execution. Once we have stated this fact, we should now wonder what the actual energy consumed by smart-contracts is.

C. *Deriving energy consumption from gas consumption*

We believe that research on smart-contracts' energy efficiency presents two benefits: reducing overall blockchain energy consumption and reducing the financial cost implied by smart-contracts transaction fees. Indeed, the gas consumption of a smart-contract function call depends on its complexity. The more computation it will do, the more (financially) costly it will be. This section presents a simple way to estimate a smart-contract's energy consumption based on its gas consumption. This estimation is a two-step process. First, we need to calibrate a ”gas to power” model on a given infrastructure. Then, we can use this model to deduce the energy needed to execute smart-contracts.

In order to use this model on real Ethereum data and obtain an estimation of the energy used by real-world smart-contract executions, we deployed an Ethereum node on a server and connected this node to the public Ethereum network. We were able to download all transactions emitted and validated during one year, from September 2019 to August 2020. The resulting dataset contains over 247 million transactions, including 140 million smart-contract calls.

Our experiments use the same cluster as the one described in subsection V-B. On this cluster, two smart-contracts were deployed and executed to gather metrics to calibrate our model. These two smart-contracts consist implement a Quicksort algorithm (same one as used in subsection V-B) and a Mergesort algorithm. We evaluated the energy consumption of these two smart-contracts depending on the size of the input array. The evaluation was conducted *in situ*, by deploying the smart-contract on the Ethereum network and generating calls to the sorting algorithm with a random integer array of a given size. We collected the average power consumption of all servers during the experiment. For the Quicksort algorithm, the results are illustrated in Figure 10.

[10]In fact, the second version of Ethereum, in development, will include a Proof-of-Stake engine that does not include any mining.

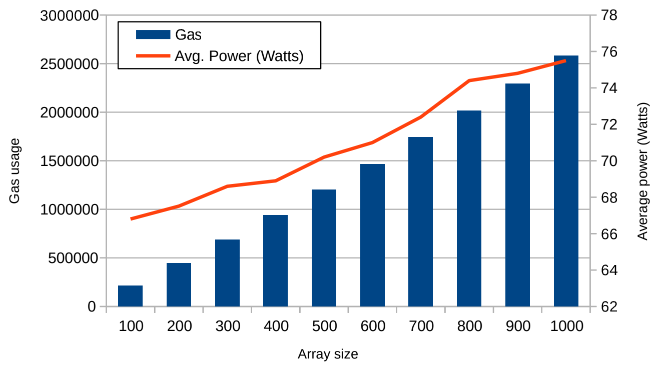

Fig. 10. Quicksort - Gas consumption and average power depending on array size input

We can notice that power and gas consumption are both correlated. Those two variables get a Pearson correlation coefficient of 0.99, indicating a linear correlation. Experiments on Mergesort lead to the same conclusion, with a coefficient of 0.93.

Using this data, we generate a linear regression model to infer the energy consumption of smart-contracts. The estimation given by the model will be tied to the infrastructure on which the model has been calibrated. However, this can give us an idea of what it would cost to run smart-contracts on this infrastructure. The model resulting from our experiments is illustrated in Figure 11. On our infrastructure, the power needed to run a smart-contract consuming X gas is given by the following function:

$$Power(X) = X * 0,0000036756792144 + 66,1136 \quad (1)$$

In our real-world year long dataset, the median value of X (the gas consumed) for smart-contract calls is 54 842, with values going from 21 051 to 12 188 440. Based on this power usage $Power(X)$ (in Watts) and the time T (in seconds) needed to run a given contract, we can compute its energy consumption (in Joules) with a function $Energy(X) = Power(X) * T$. We recall that 1 Watt equals 1 Joule per second. In our example, we run an average of 4164 smart-contracts calls in 30 minutes for each experiment for both algorithms, resulting in a 0.43 seconds contract call.

If we consider the difference in smart-contract calls' running time as negligible, we can have a rough estimation of the energy needed to run smart-contracts on our infrastructure. Estimation results are illustrated in Table III. We estimate the average energy consumption of smart-contract calls to be around 29.47 Joules. These values correspond to the energy consumed on a single Ethereum peer. As we will see in the rest of the paper, smart-contracts are executed by the entire network on each call, by each peer, which multiplies its energy consumption.

D. *Ethereum smart-contract execution model*

We have just seen that we could model Ethereum smart-contracts' energy consumption according to their gas usage.

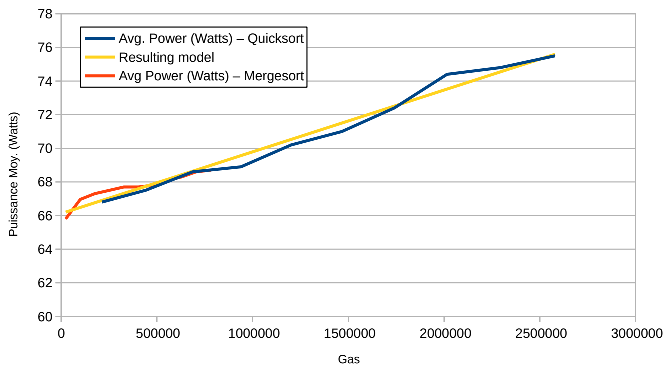

Fig. 11. Power depending on gas consumption

	Smart-contracts	Simple Txs
Min (Joules)	28,66	28,66
Max (Joules)	73,38	28,66
Average (Joules)	29,47	28,66
Standard deviation	4,49	0

TABLE III
ESTIMATED POWER USAGE FOR ETHEREUM PROCESSED SMART-CONTRACTS AND TRANSACTION OVER THE YEAR

Estimations in Table III only correspond to the energy consumption of smart-contracts on a single peer. As blockchain networks consist of large networks of peers distributed across the world, we cannot model smart-contracts' energy consumption by reasoning on a single node.

Indeed, even if transaction fees are paid only once by the caller, transactions and smart-contract calls are executed across the whole network. Figure 12 illustrates the replication of smart-contracts calls on a three nodes Ethereum Proof-of-Authority network [11]. Each of the three curves illustrates the power usage of each three servers. We emitted a call to the smart-contract sorting function. This contract call has been sent to Gros-42 only. The first spike in energy usage at 163 seconds corresponds to the execution of this call on Gros-42. This call is then transmitted to the two other nodes that will execute it. This experiment shows us that three smart-contract executions have resulted from a single call. As blockchains assume an untrusted environment, each node has to execute transactions to verify their output. As a result, the bigger the network, the more expensive each transaction and call will be.

E. *The impact of replication on smart-contracts execution cost*

Previous work in [36] has been done in order to estimate the number of peers in the Ethereum network. When this study has been written (early 2018), the authors measured 15 454 peers in a single day. To obtain this estimation, the authors have modified an existing implementation of Ethereum to build a tool named *NodeFinder*. Main modifications are that NodeFinder doesn't have a maximum peer limit (it seeks to connect to as many peers as possible) and it disconnects from

[11]Infrastructure and deployment protocol is the same than the one described in subsection V-B. Smart-contract deployed is Quicksort.

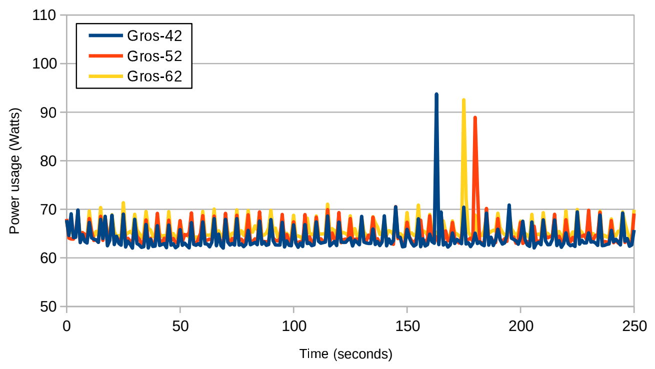

Fig. 12. Impact of smart-contract calls replication on Ethereum network energy usage

peers right after the initial Ethereum handshake (to limit the number of active connections). NodeFinder will also periodically reconnect to known peers to ensure that those are still active. At the same time, the authors reported that ethernodes.org (a website that present statistics on the Ethereum network) listed 4717 peers, meaning that NodeFinder outperformed existing methods by a factor of 2.3.

Figure 13 present the recent evolution of the number of active Ethereum nodes counted by ethernodes.org. On June 11th 2021 (at the time of writing this paper), the number of peers presented on ethernodes.org was 4036. However, we can see a sudden drop in the number of active peers in early March 2021, with a number of listed nodes going from eleven thousand to less than five thousands. We estimate that this drop may be linked with Ethereum *Berlin* update, which occurred at the same time. We can't be sure if this update has affected the number of active peers (e.g., nodes that refused the update went to another network or were shut down by their owners) or if it has affected the number of peers listed by the website (e.g., due to a change in the node discovery protocol). However, we can estimate that the real number of active peers may either be equal or higher than 4036.

Figure 14 gives a rough estimation on the energy consumed by an average contract call depending on the number of nodes in the network. As each contract call is executed on every node in the network (see subsection V-D), we projected an energy consumption what would be linear with the number of nodes.

Considering that the number of peers in Ethereum network was around four thousand – and with the strong assumption that the smart-contracts are executed in the same type of infrastructure as ours – we can consider that the energy consumed by an average smart-contract call ranges from 29.47 (estimation on single node) to 1×10^5 Joules (replication of the contract call on each node). Multiplied by the 140 143 091 contract calls emitted over the year [12], it results in a global energy consumption of 1.67×10^{13} Joules. Prior to the sudden drop in early March 2021, we can estimate the energy consumption of a single average contract call to be around 3×10^5 Joules (replication on eleven thousand nodes).

[12]As detailed in subsection V-C, around 140 million contract calls were found in the one-year dataset we extracted

F. Limitations

Our model is subject to two limitations. First, it only includes the execution cost of smart-contract calls, excluding the network communication energy cost. Second, it does not take into account the long-lasting energy cost of blockchain transactions. As new nodes will join the blockchain network, some may replay past transactions during their synchronization to ensure that the data they download from other peers is correct. To stay simple, our model only consider the execution of new transactions in active nodes. However, more complex models to be helpful to give more accurate estimation on the energy consumption of smart-contracts.

VI. Related work

A. Performance testing of blockchains

Blockbench [34] is an academic tool that aims to analyze private blockchains. Blockbench uses two workloads, YCSB [37] and Smallbank [38], to quantify the transaction rate, latency, scalability and failure resistance of three blockchain technologies (two implementations of Ethereum [39][40] and one of Hyperledger [41]). Deployment of blockchains to be tested is managed through bash scripts that do not offer abstractions over the targeted testbed. On the contrary, playbooks written in Ansible and deployed with BCTMark can be used to deploy an arbitrary number of peers on any testbed that supports SSH connections. BCTMark also provides the same abstraction over network, enabling scientists to express easily network constraints and topology. While Blockbench collects metrics about performances (latency and throughput), BCTMark can also collect both system metrics like CPU, memory or disk usage (important to consider the overall footprint of blockchain technologies) and functional metrics (e.g., number of connected peers). Finally, Blockbench only targets private blockchain whereas BCTMark also target public blockchains (as demonstrated in the experiments).

Hyperledger Caliper [42] is a tool maintained by the Hyperledger Foundation. Hyperledger Caliper is well integrated with Hyperledger products and offers a complete lifecycle similar to the one we introduced in subsection III-A. Hyperledger Caliper can monitor blockchain performances but also server metrics through Prometheus [43]. Unfortunately, it does not seem to include any network emulation, crucial for studies on the impact of network failure or latency on a blockchain. Moreover, Hyperledger Caliper does not seem to include any functionality for resources reservation on scientific testbeds like Grid'5000.

Boston Blockchain Benchmark (BBB) [44] is another benchmarking tool that enables the deployment and evaluation of blockchains on emulated hardware. BBB uses Mininet [45] to deploy the SUT. It enables a fine control on the hardware and network as both are emulated. Like BCTMark, BBB enables fine-grained network configuration with parameters like latency, packet loss and bandwidth limitations. The main

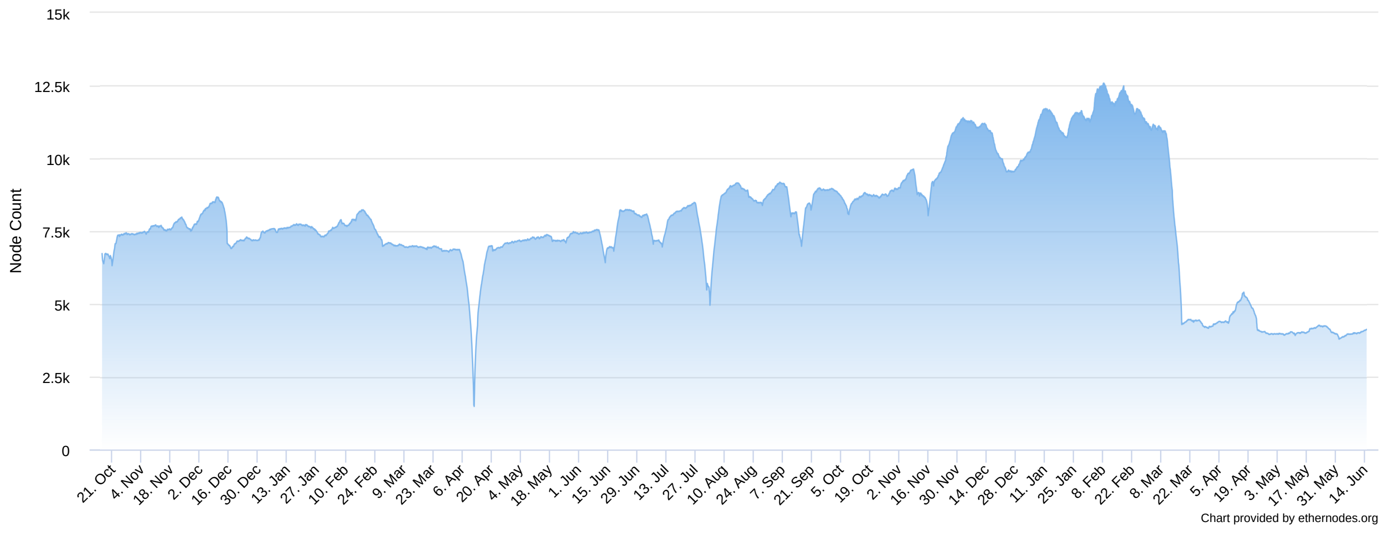

Fig. 13. Evolution of the number of Ethereum nodes listed by ethernodes.org, from October 2020 to June 2021 (source: ethernodes.org)

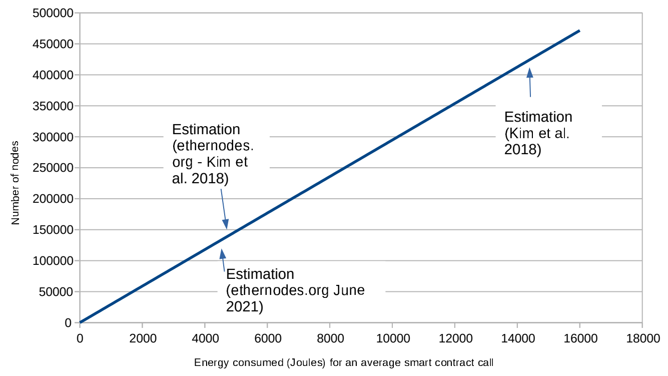

Fig. 14. Global energy consumed by a smart-contract call depending on the number of nodes in the network

difference with BCTMark is that BBB deploys the whole blockchain network on a single server: each peer is deployed on its own process. While this may simplify experiments, hardware emulation can limit some experiments. For instance, evaluations on energy consumption like we have done in section V needs proper energy monitoring. Having the whole blockchain network on a single server makes fine-grained energy monitoring more difficult.

DAGbench [46] is a benchmarking tool targeting *Directed Acyclic Graph* (DAG) based distributed ledgers. Authors have written their framework from scratch in Javascript. DAGBench currently support three DAG-based blockchains (but could be extended for more): IOTA [47], Nano [48] and Byteball [49]. Authors compare those three technologies on several criteria like latency, throughput and CPU consumption. The authors have implemented two workloads: one that transfer arbitrary data between two accounts and one that read data on the blockchain.

In summary, BCTMark seeks to be a general-purpose blockchain benchmarking network that allows experiments to be deployed on real hardware and emulated network to be as close as possible to a real-world deployment. Main differences in terms of functionalities between those tools and BCTMark are summarized in Table IV[13].

B. *Analysing uses and performances of smart-contracts*

Several papers have studied the use and performances of smart-contracts. However, to the best of our knowledge, this paper is the first attempt to measure and model the energy consumption of smart-contracts.

Pinna et al. [50] proposed an empirical study on 10 000 Ethereum smart-contracts. This study is made from a software engineering point of view, detailing metrics like EVM compiler version, number of lines of codes, number of declared contracts, and functions... The authors have reported eight key observations. For instance, newly deployed contracts often tend to use the last version of their programming language. This study provides information on how contracts are developed but does not include any energy footprint analysis.

Chen et al. [51] discuss the financial cost of under-optimized smart-contracts in Ethereum. They identify several gas-costly patterns (e.g., the presence of ”dead code”) and quantify, in a set of smart-contracts, the presence of those patterns. Based on those patterns, they proposed a tool (*GASPER*; **GAS**-costly **P**atterns check**ER)** that can discover those gas-costly patterns. We believe that such an approach could help to reduce the energy consumption of smart-contract calls. Unfortunately, the authors do not quantify the overhead in gas of those patterns, preventing the estimation of potential benefits based on our model.

Feist et al. [52] proposed a static analysis framework for smart-contracts named *Slither*. The proposed framework converts smart-contracts written in Solidity to an intermediate representation better suited for analysis. Slither has been conceived for four use cases: 1) vulnerabilities detection, 2)

[13]*SUT = System Under Test*

	Blockbench	**Hyperledger Caliper**	**BBB**	**DAGBench**	**BCTMark**
Targeted systems	Private Blockchains	Mainly Hyperledger systems	Private blockchains	DAG-Based blockchains	Every blockchain
Deployment management	No	Yes	Yes (but on a single server only)	No	Yes
Network emulation abstraction	No	No	Yes	No	Yes
Portability to new testbeds	N/A (do not manage deployment)	Yes (but no management of resources reservation)	Yes, as hardware is emulated (as long as Mininet can be installed)	N/A (do not manage deployment)	Yes (as long as testbed has SSH)
Metric collections	Yes (SUT)	Yes (SUT + testbed)	Yes (SUT)	Yes (SUT + testbed)	Yes (SUT + testbed)

TABLE IV
COMPARISON OF FUNCTIONALITIES WITH THE STATE OF THE ART

code optimization, 3) code understanding for developers, 4) assistance with code review through a provided API. The authors have compared the performances of each component of Slither with state-of-the-art tools, demonstrating good performances. Slither does not include any notion related to energy consumption at the moment.

A more comprehensive list of smart-contract analysis tools has been compiled by Di Angelo and Salzer [53]. The authors have illustrated the key features of several academic and non-academic tools and their differences. The lack of any notion of energy seems to indicate that this issue has not been tackled before.

VII. FUTURE WORK

The goal of this paper is to illustrate how BCTMark could be used to measure the energy consumption of smart-contracts. To first illustrate BCTMark's capacities, we implemented experiments on two different testbeds and three different blockchains. In the future, we would like to integrate more blockchains. Layer-two blockchain solutions (e.g., the Lightning network [13]) are presented as a potential solution to reduce the transaction replication cost of blockchains. Integrating those novel systems and measuring their impact on blockchain network performances could lead to new research ideas.

Regarding the proposed model to evaluate the energy consumption of smart-contracts, we would like to complexify this model by taking into account the energy consumed by new nodes joining the network (and replaying the blockchain history). Those model could be integrated to existing blockchain simulators. Moreover, as mentioned before, so-called layer-two blockchain systems aim to reduce the amount of information and/or computation done on the blockchain. Incorporating those new systems in our experiments could help to better understand how those can help to reduce the energy footprint of smart-contracts.

VIII. CONCLUSION

In this paper, we introduce BCTMark, a framework for benchmarking blockchains. Existing tools, while promising, do not include important aspects of reproducible experiments on blockchain systems like network emulation or reproducible deployment. BCTMark aims to empower developers and researchers to create reproducible experiments on blockchain performances. For this purpose, BCTMark provides abstractions over testbeds and network. Furthermore, to facilitate the development of benchmarks, BCTMark includes functionalities like load generation and metrics collection. To illustrate BCTMark's functionalities, we have run three experiments on three blockchains (Ethereum Ethash *vs* Clique and Hyperledger Fabric) and two testbeds (one Grid'5000 cluster and one Raspberry Pi cluster).

To provide a better illustration of potential usages of BCTMark, we have also studied, in section V, the energy consumption of smart-contracts on Ethereum. Based on their cost in gas, we measured and modeled their energy footprint. We then used this model to obtain an order of the energy consumed by Ethereum smart-contracts over of one year (based on real-world data). Key results are that despite the low cost of a single contract call, replicating each contract call over the network generates an important footprint. We estimate the energy consumed by a single contract call to be around 1×10^5 to 3×10^5 Joules, depending on the hypothesis on the number of Ethereum nodes. Thus, over one year, smart-contracts execution would have consumed around 1.67×10^{13} Joules.

BCTMark's code is open-source and accessible here: https://gitlab.inria.fr/dsaingre/bctmark.

ACKNOWLEDGMENT

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

This paper has been financed by the *HYDDA FSN* project.

REFERENCES

[1] Steem, "Steem - An incentivized, blockchain-based, public content platform," https://steem.com/steem-whitepaper.pdf, 2016, whitepaper.
[2] "E-Estonia," https://e-estonia.com/.
[3] P. Labs, "Filecoin: A Decentralized Storage Network," https://filecoin.io/filecoin.pdf, 2017, whitepaper.
[4] I. Storj Labs, "Storj: A Decentralized cloud storage network framework," https://storj.io/storjv3.pdf, 2018, whitepaper.
[5] M. Mylrea and S. N. G. Gourisetti, "Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security," in *2017 Resilience Week (RWS)*. IEEE, 2017, pp. 18–23.

[6] E. Mengelkamp, J. Gärttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, "Designing microgrid energy markets: A case study: The brooklyn microgrid," *Applied Energy*, vol. 210, pp. 870–880, 2018.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 45–59.

[8] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[9] H. Vranken, "Sustainability of bitcoin and blockchains," *Current opinion in environmental sustainability*, vol. 28, pp. 1–9, 2017.

[10] K. J. O'Dwyer and D. Malone. (2014) Bitcoin mining and its energy footprint.

[11] F. Saleh, "Blockchain without waste: Proof-of-stake," *Available at SSRN 3183935*, 2019.

[12] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," 2018.

[13] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[14] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[15] B. Smaalders, "Performance anti-patterns," *ACM Queue*, vol. 4, no. 1, pp. 44–50, 2006.

[16] "Grafana - the open observable platform," https://grafana.com/, accessed: 2019-12-6.

[17] D. Saingre, T. Ledoux, and J.-M. Menaud, "Bctmark: a framework for benchmarking blockchain technologies," in *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2020, pp. 1–8.

[18] "PoET 1.0 Specification," https://sawtooth.hyperledger.org/docs.

[19] E. Foundation, "A Next-Generation Smart Contract and Decentralized Application Platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2013, whitepaper.

[20] V. Saraph and M. Herlihy, "An empirical study of speculative concurrency in ethereum smart contracts," *arXiv preprint arXiv:1901.01376*, 2019.

[21] W. Gavin, "Ethereum: a secure decentralised generalised transaction ledger," https://ethereum.github.io/yellowpaper/paper.pdf, 2020, yellow-paper.

[22] "Ethereum Wiki - Ethash," https://github.com/ethereum/wiki/wiki/Ethash.

[23] C. Lal and D. Marijan, "Blockchain testing: Challenges, techniques, and research directions," *arXiv preprint arXiv:2103.10074*, 2021.

[24] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A detailed and real-time performance monitoring framework for blockchain systems," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2018, pp. 134–143.

[25] M. Alharby and A. van Moorsel, "Blocksim: a simulation framework for blockchain systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 135–138, 2019.

[26] M. Zander, T. Waite, and D. Harz, "Dagsim: Simulation of dag-based distributed ledger protocols," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 118–121, 2019.

[27] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "Verx: Safety verification of smart contracts," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1661–1677.

[28] Enoslib, "Enoslib : A framework to build experimental frameworks on various platforms," https://github.com/BeyondTheClouds/enoslib, github repository.

[29] Ansible, "Ansible is Simple IT Automation," https://www.ansible.com/.

[30] "Telegraf - the plugin-driven server agent for collecting and reporting metrics." https://github.com/influxdata/influxdb, accessed: 2019-12-6.

[31] "Influxdb - scalable datastore for metrics, events, and real-time analytics," https://github.com/influxdata/telegraf, accessed: 2019-12-6.

[32] "Locust - a modern load testing framework," https://locust.io/, accessed: 2019-12-6.

[33] J. Pastor and J. M. Menaud, "Seduce: a testbed for research on thermal and power management in datacenters," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2018, pp. 1–6.

[34] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1085–1100.

[35] H. McCook, "An order-of-magnitude estimate of the relative sustainability of the bitcoin network," *A critical assessment of the Bitcoin mining industry, gold production industry, the legacy banking system, and the production of physical currency*, vol. 2, p. 25, 2014.

[36] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 91–104.

[37] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 143–154.

[38] M. J. Cahill, U. Röhm, and A. D. Fekete, "Serializable isolation for snapshot databases," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 4, p. 20, 2009.

[39] "Go-ethereum - official go implementation of the ethereum protocol," https://github.com/ethereum/go-ethereum, accessed: 2019-12-6.

[40] "Parity - fast and feature-rich multi-network ethereum client." https://github.com/paritytech/parity-ethereum, accessed: 2019-12-6.

[41] "Hyperledger fabric," https://github.com/hyperledger/fabric, accessed: 2019-12-6.

[42] "Hyperledger caliper," https://www.hyperledger.org/projects/caliper, accessed: 2019-12-6.

[43] "Prometheus - from metrics to insight," https://prometheus.io/, accessed: 2019-12-6.

[44] H. Pan, X. Duan, Y. Wu, L. Tseng, M. Aloqaily, and A. Boukerche, "Bbb: A lightweight approach to evaluate private blockchains in clouds," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[45] "Mininet: An instant virtual network on your laptop (or other pc)," http://mininet.org/, accessed: 2021-06-15.

[46] Z. Dong, E. Zheng, Y. Choon, and A. Y. Zomaya, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 264–271.

[47] S. Popov, "The tangle."

[48] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 2018.

[49] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," *URL https://byteball. org/Byteball. pdf*, 2016.

[50] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, "A massive analysis of ethereum smart contracts empirical study and code metrics," *IEEE Access*, vol. 7, pp. 78 194–78 213, 2019.

[51] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 442–446.

[52] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 8–15.

[53] M. Di Angelo and G. Salzer, "A survey of tools for analyzing ethereum smart contracts," in *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*. IEEE, 2019, pp. 69–78.